ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO EN INFORMÁTICA

Título del proyecto:

ANÁLISIS COMPUTACIONAL DE LA TRANSCRIPCIÓN DEL ADN

DOCUMENTO

Iñaki Etxeberria Ruesta

Tutores: José Javier Astrain

Raúl Castanera Andrés

Pamplona, 25 de abril de 2017

Índice general

Capítulo 1

Introducción

En el año 2001, se consiguió secuenciar el genoma humano, aunque el honor de ser el primero en ser secuenciado lo tiene el organismo bacteriófago Phi-X174, en 1975. Secuenciar el genoma humano en aquel momento, tuvo un coste superior a los 3.000 millones de dólares americanos y hoy en día el coste se ha reducido a alrededor de 1.000 dólares. Este abaratamiento en el coste ha supuesto un auténtico boom en la genómica, sobre todo en la última década, donde se han secuenciado más de 100.000 genomas.

La informática iba a estar estrechamente ligada a la genética, por la ingente cantidad de datos que se deben manejar. Para hacernos una idea, la secuencia genética humana ocupa unos 750MB. Sin embargo, la información contenida en la secuencia completa, no se puede leer como si de literatura se tratase. Se debe hacer un análisis exhaustivo de toda la secuencia genética, localizar las regiones que sean de interés, conocer después la función de esas regiones, etc. Inevitablemente, la bioinformática se convertiría en una rama de estudio más de las ciencias.

Desde aquel primer ser que fue secuenciado y especialmente en la última década, se han secuenciado los genomas de más de 100.000 seres y su la vez se ha ido desarrollando cada vez más la bioinformática, considerando que en la actualidad está en pleno auge.

1.1. Antecedentes

El proyecto tiene su origen en la necesidad del Departamento de Genética de la Universidad Pública de Navarra (UPNA) de analizar la secuencia genética de los organismos con los que trabajan actualmente.

Mi toma de contacto con este grupo se remonta a una asignatura que cursé en 2011-2012, con el título: QUÉ DEBE USTED SABER PARA ENTENDER LA HERENCIA GENÉTICA. Esta asignatura aumentó mi interés por la genética y al empezar a plantear mi proyecto de fin de carrera en el segundo ciclo de Ingeniería Informática, me puse en contacto con la Doctora Lucía Ramírez Nasto, quien fuera mi profesora en la asignatura

mencionada. Me comentó el trabajo que se realizaba en su grupo y comencé a trabajar con Fran Naranjo, antiguo estudiante de Ingeniería Agrónoma de la Universidad. Fue quien inició al grupo en el área de la bioinformática y quien co-turorizara mi proyecto desde el inicio. Más tarde, Raúl Castanera asumiría el papel de co-tutor en sustitución de Fran.

Este trabajo viene a apoyar la labor realizada por este grupo, analizando y obteniendo datos de un hongo con el que realizan experimentos y del que tratan de conocer cuáles son sus características genéticas heredadas.

Este grupo de Genética y Microbiología, realiza diferentes estudios con el organismo *Pleurotus ostreatus*. Se trata de un hongo muy utilizado por su interés alimentario y biotecnológico. En la industria alimentaria es muy apreciado y consumido, conocido con el nombre de *Seta de ostra*. En el ámbito biotecnológico, produce encimas de gran interés: degradadoras de substancias, hidrofobinas para recubrimientos, producción de bioetanol, etc. Es por ello, que el estudio de este organismo tiene una gran repercusión y el tratamiento de la información que se genera en su estudio, parte que nos incumbe, requiere de un especial cuidado. El objetivo de este proyecto trata de determinar en un gen, cuál de sus progenitores es el dominante, mediante el análisis de su genoma. Por ejemplo, en el gen que controla la velocidad de crecimiento del organismo, conocer si está más presente el gen de uno u otro progenitor. Al conocer de antemano las características génicas de ambos progenitores, podemos saber qué característica es la predominante en su descendencia.

1.2. Bases genéticas

Antes de seguir, conviene explicar algunos conceptos genéticos para que la lectura sea más fácil.

Todo ser vivo está formado por al menos una célula. En este momento no nos importa si es un organismo unicelular o pluricelular, pues nos vamos a centrar en el ADN de dicho organismo y todas las células que lo forman comparten el mismo material genético. El ADN o Ácido DesoxirriboNucléico es una cadena de nucleótidos que codifican todas y cada una de las funciones biológicas de los seres vivos. Se trata de una receta donde está escrito, cómo debe ser la producción de melanina de la piel, formación de tejidos, etc. y es la herencia que recibimos de nuestro padre y nuestra madre, en el caso de la raza humana. En el hongo que nos ocupa, el ADN también contiene información sobre con qué rapidez y en qué cantidad se reproduce el hongo, así como también sobre la producción de determinadas proteínas que pueden ser de interés para usos industriales, entre otra mucha información.

En todos los casos, el ADN se representa como una secuencia de cuatro letras: A,C,G y T. Estas letras corresponden a las bases nitrogenadas que hay en cada nucleótido y son: Adenina, Citosina, Guanina y Timina. En este proyecto, las secuencias con las que trabajamos se componen precisamente de estas letras.

El ARN es muy parecido al ADN. Sus siglas se corresponden con Ácido RiboNucléico y podemos considerarlo como una traducción del ADN. En el ADN está la receta y el ARN es una traducción de esa receta, que se usará para producir las proteínas que regulan nuestras funciones biológicas. También se trata de una cadena de nucleótidos, aunque no es tan larga como el ADN ni tampoco está formada por las mismas bases nitrogenadas. En el caso del ARN la Timina se sustituye por Uracilo, por lo que una secuencia de ARN estaría formada por las letras A,C,G y U.

Un gen es la instrucción que codifica la producción de una proteína con una función concreta, como puede ser el color de nuestros ojos, o el número de descendientes que tendrá nuestro hongo. Cada gen está formado por una cadena de nucleótidos y lo veremos representado como una secuencia de A,C,G,T. Su longitud varía de unos organismos a otros y dependiendo de qué gen se trate, aunque en nuestro hongo, los genes tienen una longitud aproximada de entre 50 y 150 nucleótidos.

A la hora de leer un gen, este debe transcribirse. En resumen, se transcribe la cadena de ADN a ARN y de esta última se obtendrán las proteínas que están codificadas en el ADN. La secuencia de letras, se lee en grupos de tres, en lo que se conoce como codón. Cada codón tiene asociado un aminoácido concreto (algunos codones tienen el mismo aminoácido asociado) y al unir estos aminoácidos se forma la proteína que estaba codificada en el ADN.

En el caso de los organismos diploides, estos heredan los genes de ambos progenitores. Sin embargo uno de los dos genes puede expresarse más que el otro. Veamos un ejemplo más cercano: mi padre tienes ojos de color azul y mi madre de color marrón. En mi caso, los genes que contienen las instrucciones que colorearán mis ojos, tienen la información para dar color azul como también marrón. Dependiendo de cuántas veces se transcriba el gen de marrón o azul, mis ojos serán más de un color o del otro, por la sencilla razón de que uno de los genes se expresa más que el otro.

Obviamente, las secuencias del gen de ojos azules y ojos marrones no van a ser idénticas, la secuencia de nucleótidos será diferente. Además de esas diferencias, existen otras conocidas como SNP o *Single Nucleotide Polymorphism*, que se trata de la variación de una sola letra en la secuencia del gen. Una marca característica que diferencia la secuencia genética de un progenitor de la del otro, aunque quizás en la expresión, ambas secuencias sean idénticas.

1.3. Objetivo

El grupo de genética de la UPNA viene trabajando con este organismo desde hace más de 20 años. Hace un tiempo, vieron la necesidad de conocer la expresión alélica de un individuo concreto, sus características genéticas al fin y al cabo, pero no disponían de ninguna herramienta. Los objetivos de este trabajo vienen a cubrir estas nuevas necesidades.

Durante el trabajo del equipo, se suele secuenciar el transcriptoma de los individuos en estudio. Hace unos años, comenzaron a desarrollar un programa que analizaba los organismos en busca de ciertas características que les fueran de interés. Sin embargo a medida que avanzaba el tiempo, el aumento de datos obtenidos en los estudios supuso un problema, tanto en su tratamiento como en su visualización y comprensión. Por una parte, hay muchos datos que almacenar, procesar y comparar. Típicamente el almacenamiento no es un gran problema, por el bajo coste que supone. Sin embargo, procesar y comparar muchos datos supone un problema mayor, pues hay que hacerlo en unos márgenes de tiempo razonables, más bien cortos. Aquí es determinante una buena programación y aprovechar los recursos que estén al alcance de la investigación.

Por otro lado, no todas las personas en los grupos de investigación tienen por qué tener conocimientos informáticos, por lo que el uso que pueden hacer de las computadoras es limitado.

Mi objetivo es desarrollar una herramienta de análisis para obtener los resultados de la expresión alélica diferencial. Su uso deberá ser sencillo, la ejecución rápida y deberá mostrar resultados que puedan interpretarse fácilmente.

1.4. Estado del arte

Hasta ahora, para poder saber si un individuo había heredado un gen de uno u otro progenitor, el estudio se limitaba a ese único gen, tanto por los pocos datos que se podían obtener como por la complejidad de obtener resultados fiables de ellos.

La técnica que se utilizaba para averiguar el origen de un gen, se trataba de marcar con un elemento fluorescente un gen concreto y según la cantidad de fluorescencia medida, se infería la expresión de ese gen. Esta técnica tenía el limitante de hacerlo gen a gen, preparar el estudio y el tiempo de espera para medir la fluorescencia, además de no se muy exacto.

No es hasta 2005 cuando se secuencia el genoma completo del *Pleurotus ostreatus*. Pasarán unos años hasta que el coste de secuenciación sea asequible y asumible. Cuando se decidieron a obtener los transcriptomas de los individuos con los que realizaban estudios, se encontraron con un mar de datos, donde ya disponían de la expresión de todos los genes de una sola vez, de forma rápida y sencilla, pero cuya interpretación no era asumible.

En general los análisis de expresión alélica, se limitaban a genes o regiones muy concreta, mientras que un análisis global no era muy común.

1.5. Propuesta

En el ámbito de la genética se trabaja con una cantidad ingente de datos y no es factible analizar estos sin la ayuda de computadoras. Nuestro grupo de investigación se encuentra en la misma tesitura. A modo de ejemplo: nada más comenzar el proyecto, tuve que analizar un par de ficheros por cada uno de los dos progenitores de un hongo. La suma de los cuatro ficheros totaliza más de 95MB de pura letra, 1.525.918 de líneas con casi 70 millones de letras que conforman el ADN de los hongos en cuestión. Las cifras abruman, más aún, al darnos cuenta de que las secuencias genéticas no pueden leerse como si de literatura se tratarse. Ni siquiera tendría demasiado sentido leerlas de comienzo a fin de la secuencia, pues caprichos de la naturaleza, a veces la secuencia está escrita al revés (transpuesta), se parte y tiene zonas que no nos interesan en mitad de la secuencia que nos interesa (intrones) y demás curiosidades.

El factor de la gran cantidad de datos que suponen estos análisis, nos obliga a hacer uso de herramientas informáticas para su tratamiento. A mayor cantidad de datos, es de esperar mayor tiempo para tratarlos y es aquí donde entra el segundo de los factores: una buena programación de esas herramientas informáticas, lo más optimizada posible, con el objetivo de utilizar la menor cantidad de recursos, disponer de los resultados en el menor tiempo y hacerlo con la mayor fiabilidad posible. Una mala programación, puede no aprovechar todo el potencial de un ordenador y hacernos perder tiempo a la hora de obtener los esperados resultados o de poder disponer de ese ordenador para otros análisis.

A todo lo anterior, debemos sumar una representación adecuada de los datos. Ya he mencionado que el coste de analizar los datos, crece cuando estos lo hacen, pero también crece el número de resultados, por lo que debemos poner especial cuidado en cómo los representamos para que el usuario pueda hacer uso de ellos. De lo contrario, tendremos de nuevo mucha información que no podemos manejar.

Desarrollaré una herramienta que indique para cada uno de los genes del individuo, a quién de los dos progenitores se le parece más. Algo así como saber si mis ojos, pero también el resto de mis genes y sobre todo de aquellos cuya expresión no es visible, se parecen más a los de mi madre o a los de mi padre, por poner un ejemplo cercano.

Para el desarrollo de la herramienta, he elegido el lenguaje de programación Python. Los dos lenguajes más apropiados para el desarrollo son PERL y Python. En principio, cualquier lenguaje de programación sería candidato, pero estos dos son los más presentes en programas de análisis bioinformático y mi intención no es fragmentar más el código, sino intentar estandarizarlo y seguir la corriente predominante. Ambos son muy utilizados y disponen de herramientas y bibliotecas para trabajar en el ámbito de la bioinformática, una razón más para centrarme en estos dos, pues se facilitará mucho el manejo de estructuras y cadenas genéticas.

En diferentes pruebas de ejecución, PERL supera a Python en tiempo de ejecución,

siendo mucho más rápido que Python. Sin embargo, la tendencia de varias compañías está siendo la de migrar sus sistemas en PERL a Python. Por alguna razón, Python y Biopython está cada vez más presente en bioinformática y en el grupo de investigación ya han comenzado a trabajar con ese lenguaje en sustitución de PERL, donde comenzaron algunos proyectos. Lo cierto es que el hecho de que Python sea un lenguaje orientado a objetos, con una estructura de programación muy similar a C, que es un lenguaje muy extendido, facilita su mantenimiento y distribución. En palabras de Damian Conway, miembro de la comunidad PERL y defensor de la programación orientada a objetos,'El enfoque de Perl orientado a los objetos es casi excesivamente Perlish: hay demasiadas maneras de hacerlo". Esto implica que un mismo problema, se puede implementar de muy distintas formas en Perl, lo cual dificulta la comprensión de los códigos. En Python sin embargo, la forma de programar es más estándar y su código resulta más fácil de comprender, razón por la cuál es más fácil mantener el código y distribuirlo entre personas que no lo han desarrollado. Aunque no pretendo que las personas del grupo de investigación deban mantener esta herramienta que voy a desarrollar, puede que en un futuro sea necesario y les será más sencillo de hacerlo en Python que en PERL.

Otro de los aspectos que determina el desarrollo de la herramienta es el almacenamiento de los datos. El grupo de investigación viene trabajando en su mayoría con ficheros de texto plano. Se planteó crear una base de datos relacional para el almacenamiento de los datos, pero se descartó desde el primer momento. Ninguna de las herramientas que ya disponen hace uso de una base de datos, por lo que tendría que crearla desde cero. Esta herramienta, será otra más de las que ya disponen y su uso debe ser lo más parecido a sus sistemas actuales. En ningún momento se plantea este proyecto como creación o migración de sus sistemas a uno nuevo.

Capítulo 2

Análisis y especificaciones

2.1. Metodología

La metodología elegida para desarrollar la herramienta es la Programación Extrema. Es una más de las denominadas metodologías ágiles y su punto fuerte radica en adaptarse a los cambios que vayan surgiendo en el desarrollo de una herramienta, para aumentar las probabilidades de éxito en la obtención del producto final.

La programación extrema se basa en diversos valores. SIMPLICIDAD: se darán pequeños pasos hacia nuestro objetivo, corrigiendo los errores conforme vayan surgiendo; COMUNICACIÓN: toda persona implicada en el proyecto es parte de él, por lo que participará desde el diseño hasta el código, manteniendo una comunicación constante entre todas ellas; RETROALIMENTACIÓN: se harán entregas frecuentes de software funcional y se revisará su funcionamiento. En dichas revisiones, se detectarán los fallos en la implementación y se adaptará el software a las necesidades del proyecto; RESPETO: toda persona participante del proyecto merece el respeto de sus ideas; y CORAJE: debemos ser sinceros con el progreso y las previsiones del proyecto.

Trasladando estos valores a nuestro proyecto, nos va a permitir desarrollar poco a poco una herramienta que va a ir ampliando su funcionalidad. Nos vemos obligados a mantener una comunicación constante, puesto que al equipo le faltan conocimientos de informática y a mi conocimientos de genética. En las sucesivas entregas, iremos comprobando el buen funcionamiento de todas las partes de la herramienta, detectando errores de forma temprana y corrigiéndolos, asegurando así que se cumplen todas las expectativas. Y todo, reconociendo el saber que cada una de las partes aporta al proyecto: mis conocimientos de informática y los conocimientos del grupo sobre genética.

Estos valores se traducen a su vez en varias reglas. Una de las más importantes sería la del desarrollo iterativo e incremental del producto. Esto además exige que antes de ampliar el producto, se corrijan todos los fallos que se hayan detectado. Es decir, no vamos a desarrollar nuevas funciones, hasta que no hayamos corregido las que ya tenemos

implementadas. Continuamente se revisa el producto en su conjunto y se va aumentando su funcionalidad. También implica simplicidad en el código, lo cual facilitará su comprensión y mantenimiento. Al ir desarrollando el producto poco a poco, el código implementado resulta más simple, porque aborda tareas pequeñas una tras otra. La revisión continua del proyecto dota a este de más fiabilidad, pues aspectos que se hayan pasado por alto en un primer momento, saldrán a la luz en las siguientes revisiones. Implica también una integración cliente-desarrollador muy alta, en la que ambos toman parte en todas las fases que se sucedan, desde el diseño, donde se establecen los requisitos que se tengan que abordar, pasando por la implementación, explicando al cliente cómo se tratan los datos, hasta las pruebas, donde tanto el desarrollador como el cliente comprueban el buen funcionamiento del producto.

Cabe mencionar, que una de las reglas utilizadas en esta metodología, no se aplica en este proyecto y no es otra que la de desarrollo en parejas. Esta regla establece que dos personas trabajen en una misma máquina desarrollando el código, de manera que discutiendo sobre la implementación, se consiga un código optimizado y simple. En este caso, hay un solo desarrollador, por lo que no se puede formar una pareja que trabaje conjuntamente.

2.2. Requisitos funcionales

En un principio, al sistema se le requería que funcionara en cualquier PC. Sólo son necesarios los cuatro archivos iniciales que nos tienen que ser facilitados. En alguna ocasión, se dispondrán de los ficheros intermedios que se van generando en el proceso, por ello el sistema se desarrollará para el peor de los casos, que es disponer de la información más básica.

El sistema se compartimentará en secciones que den resultados con sentido para la biología. Unido a lo dicho en el párrafo anterior, si disponemos de los ficheros del segundo paso, el sistema está preparado para realizar todos los cálculos a partir de ese punto.

El modus-operandi del grupo de investigación se basa en ficheros, por lo que la información de entrada como de salida, ha de estar almacenada en ficheros de texto plano. Y en este caso, cada fase ha de dar como resultado ficheros cuyo formato y disposición de los datos sean los comúnmente aceptados.

2.3. Definición de ficheros

El formato de ficheros que usa el sistema son:

- .gtf o .gff: GFF (General Feature Format), se trata de un fichero separado por tabuladores con nueve columnas. Existen 3 versiones y en todas ellas las siete primeras

```
scaffold_1      JGI     start_codon     4684    4686    .       +       0       name "genemark.2_g"
scaffold_1      JGI     exon    5782    6014    .       +       .       name "genemark.2_g"; transcriptId 120684
scaffold_1      JGI     CDS     5782    6014    .       +       0       name "genemark.2_g"; proteinId 120684; exonNumber 2
scaffold_1      JGI     exon    6116    7087    .       +       .       name "genemark.2_g"; transcriptId 120684
scaffold_1      JGI     CDS     6116    7087    .       +       2       name "genemark.2_g"; proteinId 120684; exonNumber 3
```

Figura 2.1: Ejemplo .gff

```
scaffold_01     JGI     exon    9863    9939    .       +       .       fgenesh1_pg.01_#_1 transcriptId 1080063
scaffold_01     JGI     exon    10112   10164   .       +       .       fgenesh1_pg.01_#_1 transcriptId 1080063
scaffold_01     JGI     exon    10308   10323   .       +       .       fgenesh1_pg.01_#_1 transcriptId 1080063
scaffold_01     JGI     exon    10394   10441   .       +       .       fgenesh1_pg.01_#_1 transcriptId 1080063
```

Figura 2.2: Ejemplo .gtf

columnas siguen la misma estructura, siendo las dos últimas en las que hay pequeñas diferencias. Los campos o columnas contienen la siguiente información:

1. Secuencia: nombre de la secuencia donde se encuentra la característica
2. Fuente: nombre del programa mediante el cual se ha obtenido esta información
3. Característica: nombre de la característica (ej. exon, gene, stopcodon, etc.)
4. Inicio: inicio de la característica (1..n, offset de base 1)
5. Fin: fin de la característica (1..n, offset de base 1)
6. Score: valor numérico de la fiabilidad de los datos obtenidos de esta característica
7. Strand: sentido de la característica (+, -)
8. Marco (GFF, GFF2) / Fase (GFF3): Fase del CDS. Si el codón empieza en la 1ª, 2ª o 3ª base.
9. Atributos: toda la información sobre la característica

Nos encontramos con algún fichero .gtf (Gene Transfer Format). En realidad se trata de una evolución de GFF2 y a todos los efectos, para el uso que hacemos, lo consideramos como un .gff salvo por un detalle: en .gff, en el archivo .fasta asociado, la secuencia o scaffold se define como *chr* mientras que en GTF la encontramos como *scaffold*. En ambos archivos, .gtf/.gff aparece como *scaffold*.

- .fasta: se utiliza para almacenar secuencias de nucleótidos. Su formato es muy simple: encontramos un símbolo de mayor que al inicio de la línea y seguido el nombre de la secuencia o scaffold en nuestro caso. Las siguientes líneas, hasta encontrar otra que comience por »", serán líneas de ancho fijo de letras que representan a los nucleótidos: A C G T. Cualquier otro carácter simplemente se desecha. En estos ficheros se ha de tener en cuenta que se trabaja con un offset de base 1, es decir, el primer carácter se considera la posición 1 y no la 0 como suele ser común en informática.

- .bam: se trata de un archivo binario que almacena datos de secuenciamiento. Tanto BAM como SAM contienen la misma información, pero BAM la comprime para que ocupe menos y en SAM podemos visualizarla como texto plano. Como vemos en la figura 0.3, las primeras filas vienen precedidas del símbolo "@". Estas líneas

```
@HD VN:1.5 SO:coordinate
@SQ SN:ref LN:45
r001   99 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTG *
r002    0 ref  9 30 3S6M1P1I4M *  0   0 AAAAGATAAGGATA    *
r003    0 ref  9 30 5S6M       *  0   0 GCCTAAGCTAA       * SA:Z:ref,29,-,6H5M,17,0;
r004    0 ref 16 30 6M14N5M    *  0   0 ATAGCTTCAGC       *
r003 2064 ref 29 17 6H5M       *  0   0 TAGGC             * SA:Z:ref,9,+,5S6M,30,1;
r001  147 ref 37 30 9M         =  7 -39 CAGCGGCAT         * NM:i:1
```

Figura 2.3: Ejemplo .SAM

son cabeceras, que definen como está organizada la información más abajo. No me detendré más en describir este tipo de archivo, pues no manejamos la información directamente, sino que sólo haremos uso del archivo en binario BAM junto con los ficheros que hemos creado anteriormente.

Hasta aquí, hemos podido ver cómo son y cómo se organiza la información en los archivos que usaremos como de entrada de información. Obviamente hemos de conocer su estructura y organización, pues no está en nuestras manos como se guardan los datos. Los ficheros que se vayan creando en las distintas etapas y que guarden información para la siguiente, deberán seguir también una organización estándar, sin importar que la siguiente etapa sea programada por mi o sea un programa externo que use. Veamos en cada caso cómo organizo la información:

- Formato multiFasta: consiste en un fichero de texto plano, donde la primera línea comienza por el símbolo mayor que seguido de información de un gen concreto. Aquí la información del gen puede variar, pero la básica es: nombre, scaffold, inicio del gen y fin del gen. Las siguientes líneas contienen la secuencia del gen, pero sólo la parte codificante, formada por los exones. Los intrones no se incluyen en multiFasta. Se trataría de un archivo FASTA, que en lugar de almacenar secuencias de scaffolds, almacene secuencias de genes.
- Formato BLAST: aquí se genera un fichero XML. La organización de la información es bastante clara y hasta la encontramos tabulada. Describiré la información de aquellas etiquetas que nos son útiles:
  - Iteration: engloba toda la información relativa a un gen de un progenitor, que se ha alineado con otros genes del otro progenitor.
  - Iteration_query-def: contiene la cabecera del gen que actúa como *query* en la comparación.
  - Iteration_query-len: longitud de la alineación.
  - Iteration_hits: engloba todas las partes en las que se ha podido dividir la alineación. Cabe pensar que la alineación tenga que ser de inicio a fin sin cortes, pero no nos debe preocupar esta circunstancia. Tendremos en cuenta la suma de las partes alineadas y la consideramos como una total.

- Hit_def contiene la cabecera del gen que actúa como *subject* en la comparación.
- Hit_len: indica la longitud de la alineación
- Hit_hsps: aquí es donde se engloban todas las partes en las que se ha podido dividir la alineación. Cada parte, acotada por Hsp.
- Hsp_query-from: indica la posición del gen query donde comienza el alineamiento
- Hsp_query-to: indica la posición del gen query donde acaba el alineamiento
- Hsp_align-len: indica la longitud parcial del alineamiento

Ahora encapsulado en etiquetas HSP se encuentra la información de cada una de las partes en las que puede dividirse la alineación completa.

- Hsp_qseq: muestra la secuencia del gen query implicada en el alineamiento
- Hsp_hseq: muestra la secuencia del gen subject implicada en el alineamiento
- Hsp_midline: muestra cómo se alinean, nucleótido a nucleótido, ambas secuencias: la barra vertical significa alineación, un espacio en blanco significa diferencia de nucleótidos y una barra que hay alguna inserción en la secuencia.

entre las etiquetas Iteration e Iteration se encuentran todas las alineaciones para un gen de uno de los progenitores contra el otro. En nuestro caso, hemos limitado los resultados a uno, puesto que sólo nos interesa la alineación que más se parezca y esa siempre es la primera.

- Formato VCF: Variant Call Format, es un fichero de texto que normalmente se utiliza para guardar información relativa a los SNP. Recordemos que la búsqueda de SNP es en el fondo el objetivo principal de este proyecto, por lo que almacenar esa información en un formato adecuado es imprescindible. Podríamos ser más laxos en la generación de archivos intermedios, utilizar estructuras propias o quizás más cómodas, pero en este punto, se genera un archivo final y por ello es más importante que el resto. La especificación de este formato es muy intrincada, pues se puede indicar multitud de información acerca de los SNP. En nuestro caso, primamos la concreción y claridad, por lo que guardaremos el mínimo de información indispensable para nuestro trabajo y también el mínimo requerido por las especificación. Las primeras líneas del texto son cabeceras, líneas precedidas del símbolo almohadilla "#"tras las cuales vendrán los datos de los SNP. Son tres, el número de cabeceras mínimo que hay que incluir en todo archivo VCF:

  - Formato: se indicara la versión de VCF con la que se trabaja. Según la versión, las especificaciones varían, por lo que la disposición de los datos también lo hará. La primera línea ya nos clarifica esa cuestión, para que no haya dudas desde el comienzo.
  - Ficheros implicados: este campo es más como información que necesario para el tratamiento del fichero en sí. En él suelen indicarse los nombres de los ficheros de los que se ha extraído la información para generar el VCF.

Figura 2.4: Ejemplo BLAST XML

- Cabeceras de datos: los datos de cada SNP se separan por tabuladores y aquí se indica el título de cada una de esas cabeceras. Destacar que todas las cabeceras anteriores, van precedidas por dos símbolos almohadilla y este último solo de una. De nuevo, optamos por concretar y minimizar la información que guardamos:
  - CHROM: indica el número del scaffold donde se encuentra el SNP descrito en esta línea
  - POS: indica la posición absoluta, referente al scaffold, no al gen, del SNP descrito en esta línea
  - ID: identificador del SNP descrito. En este caso hemos decidido que el identificador esté compuesto por el ID del gen, seguido de un número ordinal que indica el n-SNP encontrado en ese gen.
  - REF: nucleótido de referencia en el SNP. En este caso se trataría del nucleótido de la secuencia denominada como query, en la generación del ar-

chivo BLAST

- ALT: nucleótido alternativo en el SNP. Sería el nucleótido que hemos encontrado en la secuencia subject cuando generábamos el archivo BLAST.
- QUAL: calidad en la obtención de la información de este SNP.
- FILTER: indica si ha superado todos los filtros.

Los campos QUAL y FILTER no son relevantes para nosotros, por lo que hemos indicado en todos los casos valores por defecto. En el caso de QUAL encontramos un punto y en el caso de *FILTER* encontraremos *PASS*.

- INFO: aquí se añade la información que se desee sin ningún orden establecido. Hemos decidido guardar la información de SCORE, por si en algún momento posterior quisiéramos filtrar por la calidad de obtención del alineamiento (que no de la calidad de obtención del SNP).

2.4. Requisitos no funcionales

Como requisitos no funcionales, el equipo de investigación requiere las siguientes características:

- Que la aplicación sea fácil e intuitiva a la hora de ser usada.
- Que trabaje con ficheros de texto plano y no con bases de datos.
- Sea posible ser ejecutada en cualquier equipo

Capítulo 3

Diseño

El diseño se divide en dos grandes fases. Esta división tiene su razón principal por el uso que se le va a dar a la herramienta. En la primera de las dos fases se analizan los genomas de ambos progenitores para identificar los SNP. Podemos considerarlos como los marcadores que se verifican en todas las secuencias que queramos analizar de individuos de una misma especie. Así, sólo cuando cambiemos de especie, habrá que ejecutar esta fase de la herramienta. La segunda fase será la que más se utilice y en general hablamos de funciones que nos permiten seleccionar los marcadores que hayamos obtenido.

A su vez, cada una de las dos fases se divide en subfases en las que se procesan los datos y se guardan en ficheros, para que sean utilizados por la siguiente subfase. Gracias a que hemos dividido el diseño en subfases que tienen sentido desde un punto de vista biológico, los ficheros resultantes de cada subfase, también contienen información coherente y completa que puede ser verificada por parte del equipo.

Fase uno: obtención de los SNP

Este conjunto de funcionalidades se ejecutarán sólo cuando se secuencie el genoma de un organismo por primera vez. Cabe mencionar que esto no ocurre con frecuencia, por lo que rara vez habrá que hacer uso de ellas y en el diseño las encapsulo en la primera fase.

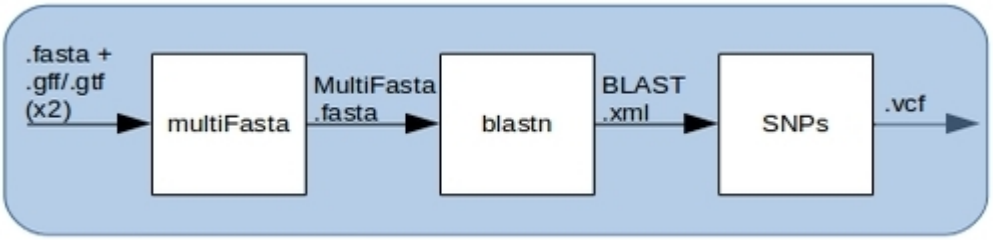


Figura 3.1: Secuencia de ejecución primera parte

En esta primera parte, esencialmente se comparan dos secuencias genéticas y se localizan los SNP.

El proceso parte de dos secuencias genéticas, de ambos progenitores y la información que nosotros queremos buscar surge de enfrentar ambas. Pero no se pueden enfrentar las secuencias completas directamente entre sí. Se han de comparar secciones de ambas secuencias que tengan algo en común: los genes. Como los genes no siempre están en la misma posición, es necesario localizarlos primero, para después localizar los SNP entre genes.

Esta primera parte, a su vez, se divide en otras tres:

1. multiFasta

   En este primer paso, obtendremos la secuencia genética de cada gen. Usando la información de los dos archivos iniciales, donde en uno tenemos el genoma completo y en el otro las posiciones de cada gen en el genoma. En el fichero .fasta encontramos la secuencia genómica completa y en el .gff/.gtf un listado de los exones que componen cada gen, con información sobre su ubicación en el genoma. Con esta información, debemos ser capaces de localizar la secuencia de cada gen en el fichero .fasta y poder agrupar la información de cada gen, junto con su secuencia.

2. BLAST

   Con la información del paso anterior, buscaremos genes homólogos. Lamentablemente, aunque sepamos dónde empieza y acaba cada gen, no tenemos tan claro cuál es su función y por tanto de qué gen se trata. En esta etapa, lo que hacemos es buscar genes homólogos y la única forma de proceder es hacer lo que se conoce como BLAST: comparar todos y cada uno de los genes de un progenitor, contra los del otro progenitor. En este proceso, se obtienen parejas de genes, que según su secuencia genética, son los más parecidos. Para ello, haremos uso de funciones ya existentes, que comparando dos ficheros multiFasta, arrojan un listado de los genes más parecidos a un gen concreto. En este paso, se toma un gen de uno de los progenitores como referencia, o *query* y se compara con todos los genes del otro progenitor. Se elige de este segundo, el más parecido y se le denomina *subject*. Normalmente para cada *query* habrá un *subject*, pero en algún caso puede que no se encuentre ningún gen candidato a ser *subject*.

3. SNP

   Aunque hayamos localizado para cada gen *query*, un *subject*, no significa que éste sea válido. Consideraremos sólo válidas aquellas parejas de genes que sean recíprocas. Determinaremos cuáles son genes recíprocos, en el caso de que la pareja de genes sea igual cuando se toma como referencia los genes de un progenitor o del otro. Esta técnica se conoce como *búsqueda de genes ortólogos mediante BLAST recíproco*. Si se tratan de genes recíprocos, buscaremos los SNP o *Single Nucleotide Polymorphism*, que no son más que discordancias de longitud uno al comparar ambas secuencias de un gen. En un

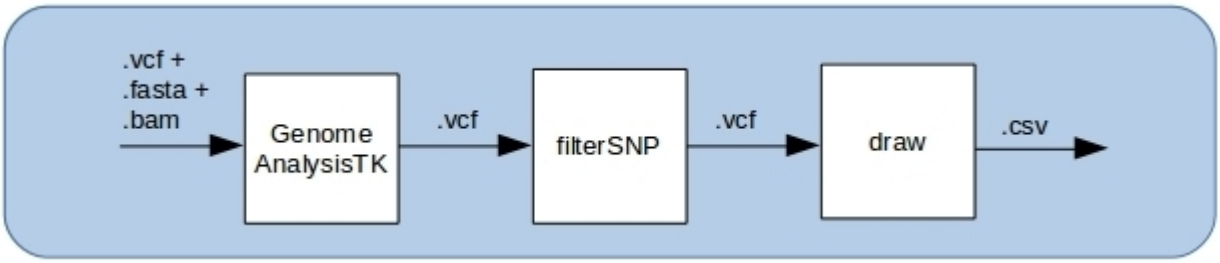


Figura 3.2: Secuencia de ejecución segunda parte

mismo gen, podemos encontrar varios SNP y se debe guardar la información de todos ellos. Utilizaremos la propia secuencia que nos devuelva la fase anterior y no la original, pues los alineamientos que se realizan en el proceso de BLAST, alteran la secuencia tal y como la encontramos en el archivo .fasta inicial.

Hasta aquí tenemos la explicación del diseño de la primera parte. Partiendo de un archivo con el genoma de un organismo y otro archivo con las posiciones donde se encuentran todos los genes conocidos para ese organismo, hemos obtenido los SNP, que nos servirán para saber si sus descendientes heredan dicho gen de un progenitor o del otro. Implícitamente encontramos la razón por la cual separamos esta parte de la siguiente: sólo se ejecuta una vez para una pareja de progenitores. Una vez hemos encontrado sus marcas características, no las volvemos a buscar.

Fase dos: contando y seleccionando SNP

En la segunda parte del sistema, se contabilizarán el número de apariciones de cada SNP en un genoma concreto. En los estudios que realiza el equipo, utilizan clones de un mismo individuo en diferentes entornos. Después se secuencian sus genomas y a la vez se obtienen las lecturas de las proteínas, que recordemos son las traducciones del genoma. A más proteínas de un gen, más veces aparecerán los SNP localizados en su región y por tanto estaremos en disposición de determinar cuántos SNP hay de un progenitor y cuántos del otro. Dicho de otra forma: si el gen de un progenitor se expresa más que el otro. Esta segunda fase se dividirá en dos partes:

1. GenomeAnalysis: se contabilizarán el número de apariciones de una proteína de un gen concreto. Hará uso de la información generada en el anterior paso, es decir, el listado SNP, así como del archivo del genoma del progenitor que se tome como referencia en el paso anterior de obtención de SNP y por supuesto, de un archivo que contenta el transcriptoma del individuo del cuál queremos saber sus características génicas.

2. filterSNP: permitirá filtrar los resultados de todos lo SNP, según los criterios del usuario, como son la distancia con otros SNP a su alrededor o algún criterio de calidad.

3. draw: se mostrarán los datos de forma visual, facilitando su comprensión y a ser posible, comparando el resultado de otros individuos.

Capítulo 4

Implementación

4.1. Entorno de desarrollo

El lenguaje de programación elegido es Python, en su version 2.7.12. El equipo de investigación ya trabaja con Python en la actualidad y no desean introducir un nuevo lenguaje de programación a su entorno. La máquina donde se ejecutará y se harán las pruebas es un portátil con procesador Intel Core i5-2410M a 2,30GHz, con 8GB de memoria RAM y como sistema operativo Ubuntu 16.04LTS

Otro lenguaje candidato para este proyecto era PERL. No se consideró otro por que estos dos son los más usados en soluciones bioinformáticas y son los que más librerías y funciones desarrolladas disponen en ese ámbito. La razón aparente de que Python sea más utilizado que PERL, es que las soluciones en PERL para un mismo problema pueden llegar a ser completamente diferentes, mientras que en Python, la propia programación orientada a objetos hace que el esquema de programación no varíe mucho. Y por tanto sea más fácil de compartir y entender estos proyectos. Todo ello a pesar de que PERL parece dar mejores resultados que Python en el tiempo de ejecución de programas. Y la diferencia puede ser bastante grande en algunos casos: hacer un BLAST le lleva 38 minutos a Python y sólo 7,28 a PERL. Ver figura 4.1.

Para el desarrollo, se han utilizado librerías para utilizar funciones específicas para trabajar con datos genéticos y son:

- blast
- samtools
- Biopython (y a su vez SciPy, NumPy, cyton)

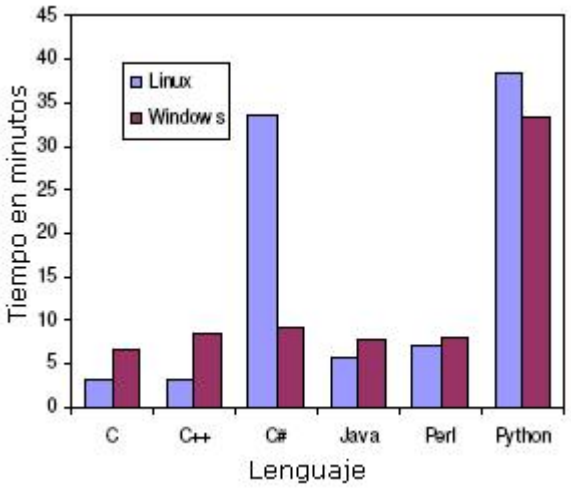


Figura 4.1: Tiempos de ejecución de BLAST en diferentes lenguajes

4.2. Desarrollo definitivo

Ahora pasaré a explicar cómo he desarrollado cada una de las partes que se especifican en el apartado anterior, haciendo especial hincapié en aquellas partes de código críticas. Se han creado scripts que se corresponden con las fases que figuran en la parte de diseño.

1. multiFASTA: Esta función recopilará la información de cada gen y la agrupará en formato multifasta. Toma como ficheros de entrada un .fasta y un .gtf o .gff. De la información contenida en estos ficheros, se obtendrá un fichero de salida .fasta, dónde se guardará la información de cada gen: identificador, scaffold, inicio, fin y su secuencia.

   El programa comienza recorriendo el fichero .gtf/.gff donde cada línea contiene un exón de un gen concreto. Los exones están agrupados por gen y ordenados en orden creciente o decreciente. La posición inicial y final del gen no la tenemos directamente, pero podemos saber cuál es la posición más baja y la más alta de todos su exones, y por tanto dónde comienza y acaba el gen. En este momento, no nos importa si el gen está invertido, sólo guardaremos su secuencia completa tal y como se encuentra en el fichero .fasta.

   Mientras leemos el fichero .gtf/.gff, cuando leamos un exon de otro gen, significa que ya hemos leído todos los del anterior y se hace una llamada a la función *writegene*, para que extraiga del archivo .fasta la secuencia desde la posición inicial hasta la final y escriba todos los datos de ese gen en el nuevo fichero multifasta.

   Uno de los problemas que surgió a la hora de localizar la secuencia en el transcriptoma, fue que había que ir leyendo el fichero y calculando en qué línea y columna iba a estar la posición de inicio del gen. Recordemos que el fichero .fasta almacena el genoma en líneas de ancho fijo. El problema se solucionó cargando el fichero entero

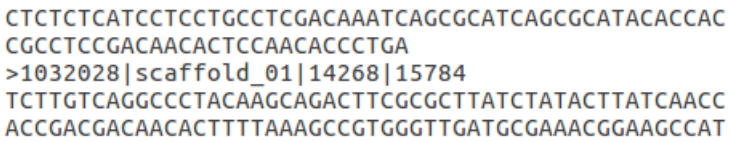


Figura 4.2: Ejemplo de fichero multiFasta

en memoria y accediendo a la posición de inicio del gen directamente, indicando la posición de inicio en la variable que contenía la secuencia completa.

Al cambiar de scaffold, se debía eliminar la secuencia hasta el primer nucleótido del nuevo scaffold y volvemos a tener los nucleótidos en la posición deseada.

2. BLAST: Conocemos cuáles son los genes y su secuencia, pero no conocemos cuál es su gen pareja en el otro progenitor. Para ello usaremos la técnica de búsqueda de genes ortólogos mediante blast recíproco. Esta técnica consiste en comparar todos y cada uno de los genes de un progenitor contra los del otro progenitor y así saber cuál es el que más se le parece. Esa comparación se debe hacer en ambos sentidos, por lo que primero uno de los progenitores se tomará como referencia y luego el otro. Al progenitor o gen de referencia nos referimos como *query* y al que se designa como gen más parecido, *subject*.

   Para este apartado, usaremos una función ya desarrollada, por la complejidad que supone y el alto coste en tiempo que supondría no programarla bien. De hecho, este apartado es el que más tiempo de esta primera parte le lleva ejecutarse. Usando esta función existente, tenemos una ventaja añadida y es que al hacer la comparación y decirnos cuál es el gen homólogo, si es que lo hay, también nos da las secuencias de ambos genes comparadas y lo que es más importante aún: los SNP (Single-nucleotide polymorphism). Aquí es donde hemos encontrado la información que nos interesa, en bruto aún, pero básicamente es lo que necesitábamos conocer para alcanzar nuestro objetivo.

   Para la comparación, necesita los archivos multifasta que hemos creado en el paso anterior y genera un archivo de salida en formato XML, donde indica el gen query, el gen subject (el más parecido), y las alineaciones de ambas secuencias enfrentadas y con unos indicadores de concordancia entre nucleótidos. La llamada a la función se hace desde Python lanzando dos procesos en paralelo.

3. SNP: En el paso anterior hemos encontrado el gen que más se le parece a cada uno, pero eso no significa que ambos genes sean compatibles, para el caso que nos ocupa. Debemos comprobar que ambos son los más parecidos entre sí de forma recíproca. Esta se trata de la parte más crucial de toda la ejecución, pues vamos a determinar con qué gen vamos a establecer la comparación para obtener los SNP, por lo tanto, si elegimos de forma errónea una pareja de genes, los resultados no serán correctos.

   El proceso comienza con los dos ficheros generados en el paso anterior, dos ficheros XML donde uno tiene como gen de referencia el de un progenitor y el otro fichero como referencia al otro progenitor. Comprobamos primero si la pareja de genes son

```
<Iteration>
  <Iteration_iter-num>750</Iteration_iter-num>
  <Iteration_query-ID>lcl|750_0</Iteration_query-ID>
  <Iteration_query-def>172505|scaffold_01|2099463|2101275</Iteration_query-def>
  <Iteration_query-len>1813</Iteration_query-len>
  <Iteration_hits>
    <Hit>
      <Hit_num>1</Hit_num>
      <Hit_id>gnl|BL_ORD_ID|899</Hit_id>
      <Hit_def>114354|scaffold_1|2435422|2437239</Hit_def>
      <Hit_accession>899</Hit_accession>
      <Hit_len>1818</Hit_len>
      <Hit_hsps>
        <Hsp>
          <Hsp_num>1</Hsp_num>
          <Hsp_bit-score>1409.95</Hsp_bit-score>
          <Hsp_score>711</Hsp_score>
          <Hsp_evalue>0</Hsp_evalue>
          <Hsp_query-from>1813</Hsp_query-from>
          <Hsp_query-to>1075</Hsp_query-to>
          <Hsp_hit-from>1</Hsp_hit-from>
          <Hsp_hit-to>738</Hsp_hit-to>
          <Hsp_query-frame>1</Hsp_query-frame>
          <Hsp_hit-frame>-1</Hsp_hit-frame>
          <Hsp_identity>733</Hsp_identity>
          <Hsp_positive>733</Hsp_positive>
          <Hsp_gaps>1</Hsp_gaps>
          <Hsp_align-len>739</Hsp_align-len>

<Hsp_qseq>AGGGGTAAAATTTGCAAGCCACAAAATGACTCATTGCATTGCACGAACTGCACGAAGAGAATGTTGGCACTTATAA
Hsp_qseq>

<Hsp_hseq>AGGGGTAAAATTTGCAAGCCACAAAATGACTCATTGCATTGCACGAACTGCACGAAGAGAATGCTGGCACTTAGAA
CACTACCGAATTTATGCGGCGGCGGCAACGACGGGTGCTGCTGGAGGCGCAGCTTCAACCGGCTTTGAAGGCTCCTCCTCCTTGGC
Hsp_hseq>
            <Hsp_midline>||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
          </Hsp>
          <Hsp>
            <Hsp_num>2</Hsp_num>
            <Hsp_bit-score>1168.1</Hsp_bit-score>
```

Figura 4.3: Ejemplo de fichero BLAST en XML

```
Query= 153284|scaffold_01|1694251|1695422
         (1172 letters)


                                                                    Score    E
Sequences producing significant alignments:                         (bits) Value

82463|scaffold_1|2857386|2858422                                      1709   0.0

>82463|scaffold_1|2857386|2858422
          Length = 1037

 Score = 1709 bits (862), Expect = 0.0
 Identities = 978/1018 (96%), Gaps = 19/1018 (1%)
 Strand = Plus / Minus


Query: 164  tcccttgtatgtcgaacgtggaaccgtatatgccgacctcacatcacatcttttggacta 223
            ||||||||||||||||||||||||||||||||| ||||| ||||||||||||||||||||
Sbjct: 1008 tcccttgtatgtcgaacgtggaaccgtatatgtcgaccgcacatcacatcttttggacta 949


Query: 224  tttctttgactggctgcgagtcggtcacctctcagctcagggccttcctccatttttactg 283
            | ||| ||||||||||||||||||||||||||||||||||| ||||||||||||||||||
Sbjct: 948  tctctctgactggctgcgagtcggtcacctctcagctcaggcccttcctccatttttactg 889
```

Figura 4.4: Ejemplo de fichero BLAST en texto plano, donde se aprecia mejor el alineamiento de los genes

recíprocos. Para ello revisamos si en ambos ficheros, el gen homólogo para uno y el otro son el mismo. En un lenguaje más técnico, comprobamos que el *query* de cada *subject* en un BLAST es el *subject* del *query* en el otro BLAST. Cuando dos genes sean recíprocos, desecharemos aquellos que tengan un porcentaje de similitud menor que 0.5. No consideramos genes homólogos, aquellos que aun siendo recíprocos, su alineamiento no se parece en al menos el 50 por ciento de su secuencia. Durante la ejecución, por un interés futuro del grupo, se diferencian aquellos que son similares en al menos un 80 por ciento de su secuencia y los que lo son hasta el 50 por ciento, pero para el resultado final, ambos resultados se mezclan y todos ellos se consideran válidos. Este valor lo calculamos obteniendo el mínimo de los porcentajes de identidad de ambos blast con respecto a ambas secuencias.

Hay otra condición para considerar válido un alineamiento de genes recíprocos y es que todas las partes en las que se divide un alineamiento, deben tener la misma dirección. En el BLAST podemos comprobar que el alineamiento se realiza por partes y en algunas ocasiones, una de esas partes se ha alineado en sentido contrario al resto, del mismo gen obviamente. Estos casos no los consideramos válidos tampoco.

Si los genes no son recíprocos, entonces se descartan directamente y no se comprueba ni similitud ni dirección de sus alineamientos.

Mientras recorremos ambos ficheros, nos encontraremos con los siguientes escenarios:

- Recíprocos: aquellos genes que en ambos blast, aparecen como pareja del otro
  - Alelos: se deben parecer en al menos el 80 % de su secuencia
  - Semialelos: ambos genes deben ser parecidos en al menos 50 % de su secuencia
  - Wrong: se parecen menos del 50 % en toda la secuencia
  - Girados: algún alineamiento está girado. Es decir, se codifica en en otro sentido al del resto de la alineación.
- No recíprocos: genes cuya pareja en uno de los dos blast es diferente
- No concordantes: genes únicos, sin pareja

Sólo con los genes que hemos dado por válidos, buscaremos los SNP para proceder a guardarlos en un fichero .vcf.

Aquí un problema supuso el acceso aleatorio que se hacía sobre el segundo fichero, lo cuál ralentizaba mucho el proceso. La solución fue crear listas con la información del BLAST, cuyo identificador es el id del gen de referencia, o gen query. Después, para conseguir los alelos, se recorren todos los genes de una de las listas y se comprueba que en la otra lista, la pareja se repite.

Los SNP (Single-nucleotide polymorphism), como indica su nombre, se considera la diferencia de un sólo nucleótido en dos secuencias comparadas. Podemos considerarlos como fallos a la hora de alinear ambas secuencias de los genes homólogos. No son fallos desde el punto de vista genético, pues ambos genes pueden ser perfectamente funcionales. Sin embargo a nuestros ojos, es una diferencia en la alineación. Si nos

```
##fileformat=VCFv4.2
##reference=PC15v2 & PC9v1
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO
9       1720149 1059265-1       G       C       .       PASS    .       2043.0
9       1720088 1059265-2       C       T       .       PASS    .       2043.0
9       1719729 1059265-3       C       T       .       PASS    .       2043.0
9       1719547 1059265-4       A       C       .       PASS    .       2043.0
```

Figura 4.5: Ejemplo de fichero VCF

encontramos dos nucleótidos o bases consecutivas que difieren con la secuencia comparada, entonces no estamos ante un SNP. Esa circunstancia no la contemplamos en este proyecto.

Este tipo de fallos, son interesantes desde el punto de vista genético, pues ayudan a identificar de qué progenitor hemos heredado ese gen en concreto.

Para buscar los SNP, usaremos uno de los datos que ya nos arrojó BLAST en su fichero. Por un lado tenemos la línea del gen query, por otro la del gen subject y la tercera es una alineación entre ambas secuencias, donde mediante barras verticales se indica si los nucleótidos son iguales o no. La búsqueda de SNP se basa en buscar un solo espacio entre dos barras verticales. Tenemos dos excepciones: al inicio y fin del alineamiento, donde la cadena a buscar es ”—ƨ ”— respectivamente. Así para cada SNP encontrado, se imprimirá una línea con toda su información asociada: scaffold, posición, snpid (geneid más un número correlativo para diferenciar los SNP de un mismo gen), nucleótido de referencia, nucleótido alternativo, calidad, filtro e información adicional. Estos tres últimos campos no son obligatorios, por lo que los valores para las columnas calidad y filtro las rellenamos con valores por defecto y en la columna de información adicional guardaremos el valor de calidad del gen, que no del SNP. Este valor nos lo proporciona el propio BLAST.

En la segunda fase, nos centramos en contabilizar el número de apariciones de un SNP concreto en cada gen. Aquí es donde finalmente veremos si se expresa más el gen de un progenitor o del otro, mediante un conteo de las apariciones de cada SNP. En esta fase, se utilizan tres ficheros como entrada de información, que listo a continuación:

- .fasta: se trata de uno de los primeros ficheros con que se inicia el programa, donde encontramos el genoma completo de uno de los progenitores. Este será el que se utilice como de referencia para ver cuántas apariciones hay. Si eligiéramos el del otro progenitor, las cuentas serían las mismas, sólo que en la columna de nucleótidos de referencia, aparecerían los alternativos
- .bam: este fichero contiene toda la información de la secuenciación de un genoma del descendiente que queramos estudiar.
- .vcf: es el fichero creado como último resultado en la primera fase. Contiene todos los SNP encontrados para cada gen.

4. GenomaAnalysisTK: con esos tres ficheros, usamos una función ya existente para el conteo, ASEReadCounter. Como salida, da un fichero de texto plano, con campos separados por tabuladores y que son:

contig	FirstSNPpos	lastSNPposition	variantID				
scaffold_01	28156	28279	153847	4	89	1	1
scaffold_01	29187	29546	1098471	5	123	1	1
scaffold_01	32806	33481	1031339	5	1083	0	1
scaffold_01	38645	38645	1110163	1	93	0	1
scaffold_01	39583	40651	19514	8	219	0	1

Figura 4.6: Vista del fichero resumen de SNP

- contig: indica el scaffold donde se encuentra el gen
- position: posición en relación al scaffold
- variantID: identificador del SNP
- refAllele: nucleótido de referencia (según el progenitor cambia).
- altAllele: nucleótido alternativo (el del otro progenitor).
- refCount: número de apariciones del nucleótido de referencia
- altCount: número de apariciones del nucleótido de referencia.
- totalCount: total del conteo
- lowMAPQDept, lowBaseQDepth, rawDepth, otherBasesherr, improperPairs no las usaremos.

5. filterSNP: esta función permite hacer una criba de todos los SNP obtenidos. Son tres los valores con los que se puede afinar la criba. Toma como fichero de entrada el fichero de texto plano generado en la fase anterior. En la llamada a esta función, el usuario debe indicar los siguientes parámetros que le ayudarán a filtrar los SNP:

   a) filterMode: 1 indica modo estricto y 2 modo relajado. En modo estricto, no se permite ninguna aparición de otro SNP antes ni después en el rango indicado por el parámetro *window*. En el modo *relajado* se permitirá que haya un solo SNP a uno de los lados.

   b) window: se indica mediante un valor, el número de nucleótidos antes y después del SNP actual en los que se comprobará si existe otro SNP.

   c) totalCount: se comprueba si el número total de apariciones de SNP, tanto de referencia como alternativos, es al menos igual a este valor. Se busca eliminar aquellos SNP cuya expresión sea muy baja.

   Como ficheros de salida encontramos dos: uno que en estructura es idéntico al original, donde aparecerán los SNP que cumplan con las condiciones que hayamos indicado en la llamada del script y otro que denomino *summary* o resumen. En este, se seleccionan los SNP que cumplen con las condiciones y se agrupan en una sola línea por gen. Los datos que se muestran en el fichero de resumen son:

   - contig: indica el scaffold donde se encuentra el gen
   - FirstSNPpos: indica la posición del primero de los SNP de dicho gen

			FAST				SLOW			
Gene id	scaffold	position	M01	M08	M20	M28	M62	M69	M82	M83
153847	scaffold_01	28279	0	0	0	0	1	1	1	1
1098471	scaffold_01	29066	3	0	3	0	3	3	3	3
1031339	scaffold_01	32898	0	0	0	0	1	1	1	1
1080067	scaffold_01	34053	0	0	0	0	1	1	1	1
1110163	scaffold_01	38645	0	0	0	0	1	1	1	1
19514	scaffold_01	39285	0	0	0	0	1	1	1	1
20823	scaffold_01	42268	0	0	0	0	1	1	1	1
1087490	scaffold_01	45038	0	0	0	0	1	1	1	1
1098479	scaffold_01	46517	0	0	0	0	1	1	1	1
17401	scaffold_01	50956	0	0	0	0	1	1	1	1
1018781	scaffold_01	54257	0	0	0	0	1	1	3	3
1098482	scaffold_01	57311	2	1	2	2	3	2	0	3
1080075	scaffold_01	63217	0	0	0	0	3	0	3	0
1110166	scaffold_01	64903	0	0	0	0	1	2	3	2
1098485	scaffold_01	67283	0	0	0	0	1	1	1	1
1032088	scaffold_01	69406	0	0	0	0	1	1	1	1
1098487	scaffold_01	71717	0	0	0	0	1	1	1	1

Figura 4.7: Muestra del fichero final con 8 conjuntos de datos

- lastSNPpos: indica la posición del último de los SNP del gen
- varianID: indica el identificador del gen
- SNPCount: su número nos dice cuántos SNP tiene el gen
- refCountSUM: suma el número de apariciones de nucleótidos de referencia en todos los SNP del gen
- altCountSUM: suma el número de apariciones de nucleótidos alternativos en todos los SNP del gen
- ourColumn: esta columna la generamos según el conteo de nucleótidos de referencia y alternativos. Será 0 si se cumple que (altCountSUM mayor que 2*refCountSUM), será 1 si (refCountSUM mayor que 2*altCountSUM) y será 2 en el resto de los casos.

Este segundo fichero de resumen es el que nos interesa. Los genes se heredan *enteros*, por lo que es de esperar que todos los SNP correspondan a un mismo progenitor y por tanto visualizar todos los SNP indicando su progenitor de procedencia, no aporta más información que si los agrupamos.

6. Draw:

   Nos encontramos en el final del proyecto y ya tenemos un fichero donde indica gen a gen, a cuál de los dos progenitores se parece más. Aunque lo interesante es poder comparar los resultados de diferentes individuos, puesto que el grupo de investigación suelen lanzar los experimentos con varios individuos a la vez. Para ello, hacemos uso de un script del que ya disponen en el equipo de investigación, que recopila el listado de genes del fichero de sumario. Con todos ellos, por cada línea mostrará los datos relativos al gen tras lo cual mostrará tantas columnas como ficheros de datos haya de

entrada, donde cada columna contendrá la última columna mencionada en el paso anterior.

Así, según el valor, las celdas se colorearán de diferente color: rojo si el valor es 0, lo cual indica que predominan el conteo alternativo, PC9 en nuestro caso. Veremos azul oscuro cuando el valor de la columna sea 1, que indica predominancia de conteos de referencia o PC15. Si el valor es 2, aparecerá en azul claro y significa que no podemos determinar cuál de los dos conteos es mayor y finalmente veremos gris cuando el valor de la celda sea 3, lo cuál nos indica que no existen datos de ese gen, para ese individuo. El resultado se puede ver en la figura 4.7

4.3. Cambios mayores

A lo largo de todo el proyecto, este ha sufrido algunos cambios y creo conveniente comentarlos.

1. Cambio formato BLAST: en un principio se desarrolló esta solución con un formato de archivo, que más tarde, se descubriría está en desuso. Aunque el cambio en el módulo de BLAST no supuso mucho esfuerzo, en SNP hubo que modificar la función que leía el fichero de entrada, para que leyera correctamente el nuevo formato en XML. No así el resto de funciones en SNP, pues no fue necesario modificar las estructuras de manejo de los datos internas.

2. Cambio formato SNP: aunque no difiere mucho el actual formato del que se ha adoptado finalmente, aprovechando el cambio en el fichero de entrada en este módulo, se cambió también el formato de salida de los datos, para adecuarlo al estándar VCF.

3. Cambio en la plataforma de ejecución: llegados a la fase de *GenomeAnalysis*, donde se contabilizan el número de SNP, vimos que era más conveniente utilizar la herramienta en el cluster que utiliza el grupo de investigación, en lugar de desarrollar un script propio y ejecutarlo en local. Se consideró como mejor solución migrar la herramienta al cluster, pues los datos de secuencias ya estaban ahí, para otros estudios. De esta manera se centraliza la ejecución de software en un único lugar, con más recursos de ejecución que los que se disponen en la universidad.

Capítulo 5

Pruebas y validación

Explicaré las pruebas realizadas por mi parte, en cada una de las fases.

- multiFasta: en este punto, se realizan dos pruebas diferentes con varios genes escogidos al azar a lo largo de todo el gen y de diferentes scaffolds. Por un lado se comprueba la corrección de los metadatos de cada gen, en concreto, sobre el inicio y final indicado de cada gen. Este punto es algo crítico, por dos razones: en los archivos .gtf/.gff no indican directamente el inicio y el fin de cada gen, sino de cada uno de los exones que lo componen. A su vez, puede que el gen este al revés, por lo que el primer valor obtenido, puede ser en realidad el final del gen. Ver comparación figuras 5.1 con 5.2.

  Después se comprobó que la secuencia extraída del fichero .fasta, sobre la secuencia del gen era la correcta, comparándola con el visor de JGI (Joint Genome Institute), donde entre otros, tienen la secuencia genómica de nuestro *Pleurotus Ostreatus*. Ver comparación de figuras 5.3 y 5.4.

- BLAST: aunque se tratan de funciones ya implementadas, compruebo algunos genes para asegurar su buen funcionamiento. Como se puede ver en la figura 5.5, el

```
scaffold_1      JGI     exon    761     1477    .       -       .       name "fgenesh1_pg.1_#_1"; transcriptId 102125
scaffold_1      JGI     CDS     761     1477    .       -       0       name "fgenesh1_pg.1_#_1"; proteinId 102125; exonNumber 2
scaffold_1      JGI     stop_codon      761     763     .       -       0       name "fgenesh1_pg.1_#_1"
scaffold_1      JGI     exon    1679    1774    .       -       .       name "fgenesh1_pg.1_#_1"; transcriptId 102125
scaffold_1      JGI     CDS     1679    1774    .       -       0       name "fgenesh1_pg.1_#_1"; proteinId 102125; exonNumber 1
scaffold_1      JGI     start_codon     1772    1774    .       -       0       name "fgenesh1_pg.1_#_1"
```

Figura 5.1: Vista fichero gff del gen 102125

```
>102125|scaffold_1|761|1774
TCACTTGTTGGTCCAGACAAAATCTGCTG
```

Figura 5.2: Metadatos del gen 102125 en fichero multifasta

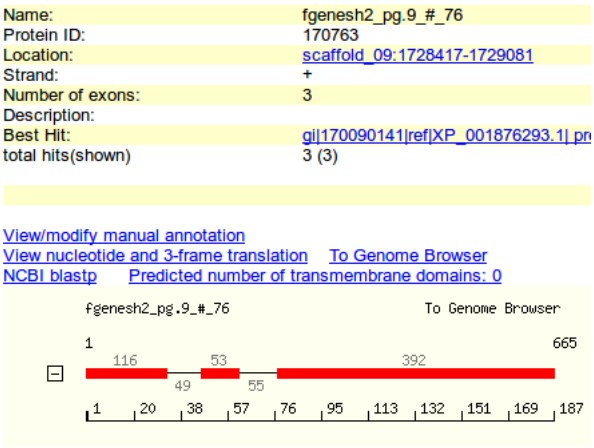


Figura 5.3: Vista resumen del gen 170763 en el visor web del JGI

Figura 5.4: Secuencia exon nº1 del gen 170763 en el visor web del JGI

alineamiento en local es el mismo que en JGI y además se muestra el mismo SNP en la 5º posición desde el inicio del alineamiento.

- SNP: para esta comprobación también selecciono varios genes al azar a lo largo del genoma. En la figura 5.6, se muestra el ejemplo del gen 1059265 de PC15. Se comprueba que para PC9 como referencia, el gen 67336 tiene como gen más parecido el mismo. Después, en un proceso más laborioso y rudimentario, se comprueban las posiciones de SNP en el fichero .xml y se comprueba que dicho SNP se incluye en el fichero de salida, con la posición así como los nucleótidos de referencia y alternativo correctos.

- filterSNP: La comprobación de este script es trivial. Como su función consiste en filtrar los resultados que tengan más cuentas que las indicadas y que los SNP estén separados por una distancia dada, basta con comprobar que no se muestran los SNP que no deben. Se toman para la comprobación el conjunto de SNP del primer y último gen del fichero. En la figura 5.7 se ven tres imágenes. La primera corresponde

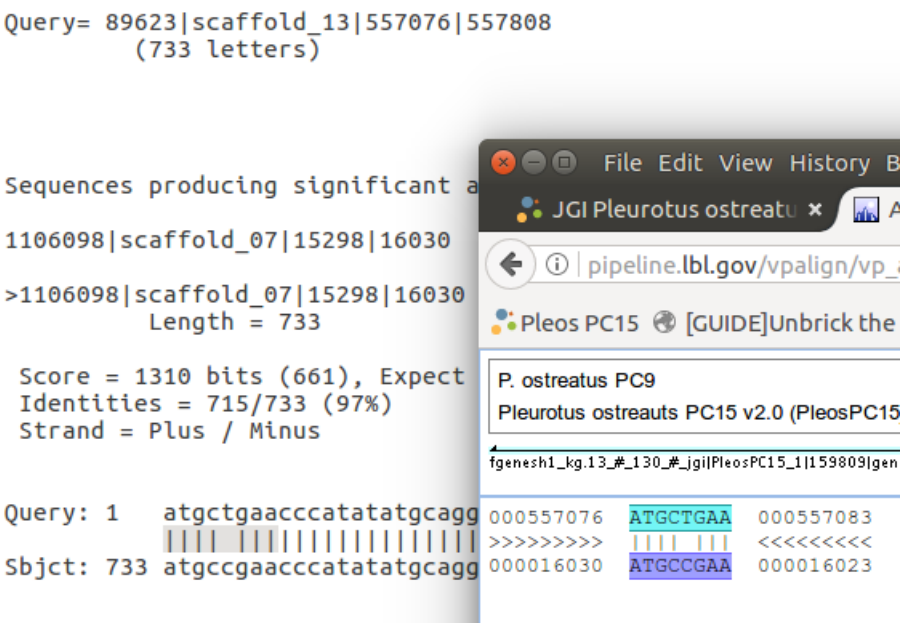


Figura 5.5: Comparación visor JGI contra BLAST local en gen 1106098

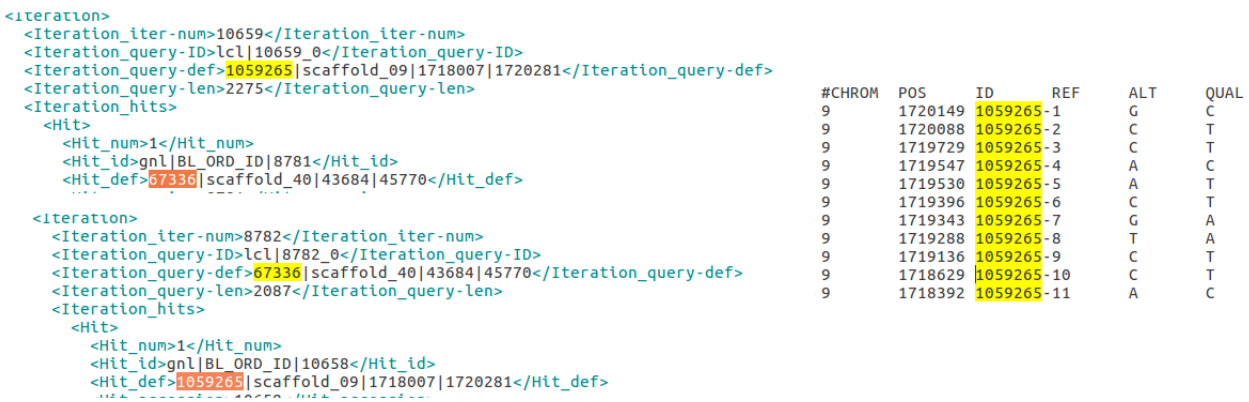
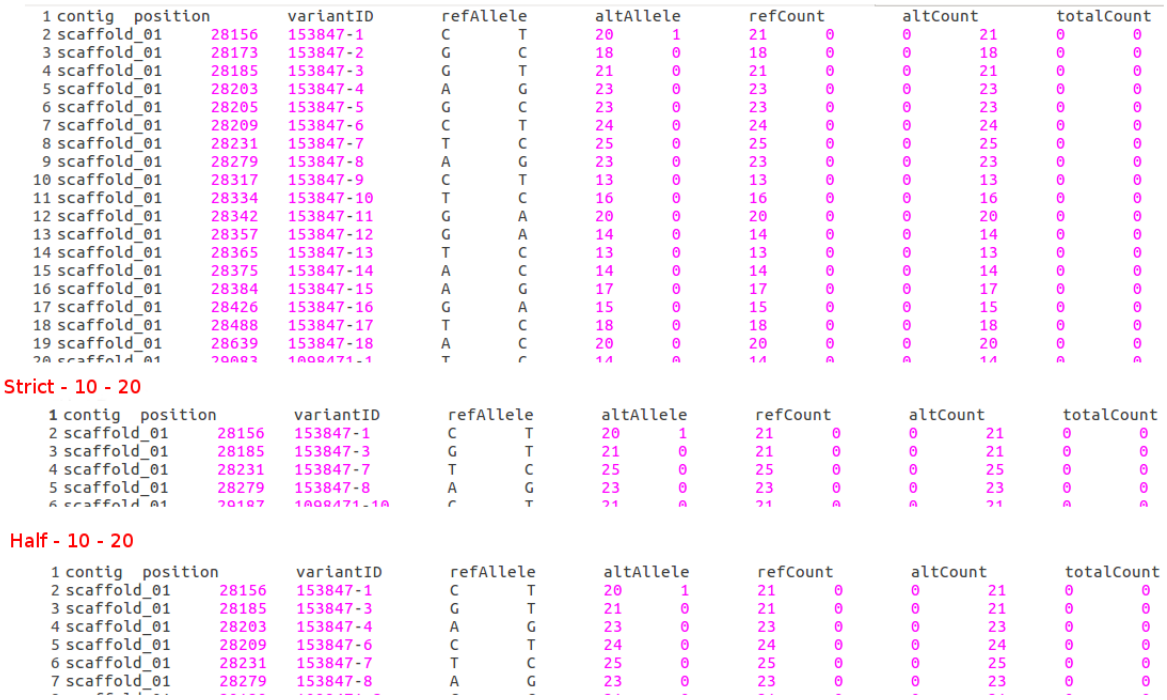


Figura 5.6: Test de BLAST



	contig	position	variantID	refAllele		altAllele		refCount		altCount		totalCount	
2	scaffold_01	28156	153847-1	C	T	20	1	21	0	0	21	0	0
3	scaffold_01	28173	153847-2	G	C	18	0	18	0	0	18	0	0
4	scaffold_01	28185	153847-3	G	T	21	0	21	0	0	21	0	0
5	scaffold_01	28203	153847-4	A	G	23	0	23	0	0	23	0	0
6	scaffold_01	28205	153847-5	G	C	23	0	23	0	0	23	0	0
7	scaffold_01	28209	153847-6	C	T	24	0	24	0	0	24	0	0
8	scaffold_01	28231	153847-7	T	C	25	0	25	0	0	25	0	0
9	scaffold_01	28279	153847-8	A	G	23	0	23	0	0	23	0	0
10	scaffold_01	28317	153847-9	C	T	13	0	13	0	0	13	0	0
11	scaffold_01	28334	153847-10	T	C	16	0	16	0	0	16	0	0
12	scaffold_01	28342	153847-11	G	A	20	0	20	0	0	20	0	0
13	scaffold_01	28357	153847-12	G	A	14	0	14	0	0	14	0	0
14	scaffold_01	28365	153847-13	T	C	13	0	13	0	0	13	0	0
15	scaffold_01	28375	153847-14	A	C	14	0	14	0	0	14	0	0
16	scaffold_01	28384	153847-15	A	G	17	0	17	0	0	17	0	0
17	scaffold_01	28426	153847-16	G	A	15	0	15	0	0	15	0	0
18	scaffold_01	28488	153847-17	T	C	18	0	18	0	0	18	0	0
19	scaffold_01	28639	153847-18	A	C	20	0	20	0	0	20	0	0

Strict - 10 - 20

	contig	position	variantID	refAllele		altAllele		refCount		altCount		totalCount	
2	scaffold_01	28156	153847-1	C	T	20	1	21	0	0	21	0	0
3	scaffold_01	28185	153847-3	G	T	21	0	21	0	0	21	0	0
4	scaffold_01	28231	153847-7	T	C	25	0	25	0	0	25	0	0
5	scaffold_01	28279	153847-8	A	G	23	0	23	0	0	23	0	0

Half - 10 - 20

	contig	position	variantID	refAllele		altAllele		refCount		altCount		totalCount	
2	scaffold_01	28156	153847-1	C	T	20	1	21	0	0	21	0	0
3	scaffold_01	28185	153847-3	G	T	21	0	21	0	0	21	0	0
4	scaffold_01	28203	153847-4	A	G	23	0	23	0	0	23	0	0
5	scaffold_01	28209	153847-6	C	T	24	0	24	0	0	24	0	0
6	scaffold_01	28231	153847-7	T	C	25	0	25	0	0	25	0	0
7	scaffold_01	28279	153847-8	A	G	23	0	23	0	0	23	0	0

Figura 5.7: Test de filterSNP ventana = 10, count = 20, modo estricto y medio respectivamente

a todos los SNP del monocarionte M83 sin filtrar. La segunda, bajo la etiqueta *Strict - 10 - 20*, los primeros SNP del gen 153847 filtrados con una ventana de 10 y cuenta de 10. Aquellos SNP que estén a una distancia menor de 10 a ambos lados o el número de cuentas sea menor que 20, no se muestran. Y bajo la etiqueta *Half - 10 - 20*, podemos comprobar que se incluyen los SNP nº 4 y 6.

- draw: este paso, no debería ser muy conflictivo, sin embargo no está exento de posibles errores, más cuando el fichero se crea con un script que dispone el equipo para juntar diferentes resultados y poder compararlos entre sí, el cuál no he diseñado yo. Aunque lo doy por bueno, conviene comprobar su funcionamiento, aplicando a los ficheros originales de salida de cuentas de SNP, las mismas operaciones que se han aplicado para después combinar esos resultados. Realizo contra-pruebas para comprobar que el script funciona correctamente al encontrar un fallo. Ver figura 5.8

Durante el desarrollo de la herramienta, he utilizado los genomas de PC15 y PC9 así como de N001 para validar el resultado. Una vez se dio por finalizado y correcto el desarrollo de la herramienta, el equipo de investigación la puso a prueba con los 8 transcriptomas de una investigación que habían realizado previamente y pudieron comprobar que la herramienta funcionaba correctamente. Los 8 transcriptomas utilizados son M01, M08, M20, M28, M62, M69, M82 y M83, los mismos que se pueden ver en la figura 5.8

				FAST				SLOW			
2	Gene id	scaffold	position	M01	M08	M20	M28	M62	M69	M82	M83
3	153847	scaffold_01	28279	0	0	0	0	1	1	1	1
4	1098471	scaffold_01	29066	3	0	3	0	3	3	3	3
5	1031339	scaffold_01	32898	0	0	0	0	1	1	1	1
6	1080067	scaffold_01	34053	0	0	0	0	1	1	1	1

1

				FAST				SLOW			
2	Gene id	scaffold	position	M01	M08	M20	M28	M62	M69	M82	M83
3	153847	scaffold_01	28279	1	0	1	0	1	1	1	1
4	1098471	scaffold_01	29066	3	0	3	0	3	3	3	3
5	1031339	scaffold_01	32898	0	0	0	0	0	1	0	1

```
1 Problem at gene 153847 at column 1
2 Problem at gene 153847 at column 3
3 Problem at gene 1031339 at column 5
4 Problem at gene 1031339 at column 7
```

Figura 5.8: Test final

5.1. Tiempos de ejecución

En este proyecto, como en muchos otros, el tiempo que se invierte en analizar los datos y mostrar los resultados ha de ser tenido muy en cuenta, pues la herramienta va a estar en uso constante y las personas que estén usándola, dependerán de los resultados de esta para continuar con su trabajo. Los resultados han de ser fiables, pero también estar disponibles en un tiempo razonable.

Las pruebas de ejecución se han realizado sobre el mismo equipo en el que se ha desarrollado la herramienta, que dispone de un procesador Intel Core i5-2410M a 2,30GHz x4, con 8GB de memoria y corriendo Ubuntu 16.04LTS, excepto la fase GenomeAnalisys, cuya ejecución y pruebas deben hacerse en el cluster.

En la siguiente tabla, se pueden comprobar los tiempos de ejecución de las diferentes fases del proyecto. En general la mejora en el tiempo de ejecución es muy notable y se debe a reorganización del código, uso de otras funciones más apropiadas o manejo de estructuras de datos gracias a Biopython.

De todas ellas, la que más cabe destacar es BLAST, sobre todo por que es la fase que más ralentiza la ejecución de la primera parte del proyecto. Se consiguió reducir a menos de la mitad el tiempo de ejecución en BLAST, aprovechando el paralelismo del que se puede hacer uso en el BLAST recíproco, pues se hacen sendos BLAST y son totalmente independientes entre sí. De los 619 segundos que tardó en la primera versión a poco más de 286 segundos.

Puede llamar la atención que en la versión final, se pase de esos 286 a 289 segundos. Sin embargo, este ligero aumento en el tiempo de ejecución es achacable al cambio en el formato de fichero de salida utilizado en BLAST. En un primer momento, se utilizó un formato de fichero en texto plano, que luego descubriría que estaba en desuso, por lo que lo modifiqué para que el fichero de salida fuera en XML, que era el otro formato disponible y el que se usa actualmente.

TOTAL secs ('.' for decimals)	v1	v2	v3
multiFastaGFF.py	13.836		
multiFastaGTF.py	1.837	0.886	0.918
multiFasta.py	1.777	0.9	
multiFasta.py	(gtf) 0.963 / (gff) 4.639		
BLAST (>v2 multiprocess)	619	295	286.832
BLAST XML	444.941	289.105	
SNP	13.424		
SNP VCFXML	28.483		
DeleteDuplicates	0.011		
GenomeAnalisys*	5100		
FilterSNP	11.437		
TOTAL TIME	5423.2 ≈1'5h		

*12 cpu + 16 GB mem / 25'5M lec

Figura 5.9: Tiempos de ejecución

Este cambio de fichero de texto plano, también tuvo su repercusión en la siguiente fase: SNP. De algo más de 13 segundos que tardaba en una primera versión a los más de 28 que tarda la versión final. Aquí había que manejar un formato de entrada distinto, pero también se modificó el formato de salida para cumplir con el formato VCF. Ambos cambios provocaron un aumento en el tiempo de ejecución.

En general, los tiempos de ejecución no variarán demasiado cuando se utilicen genomas de la especie *Pleurotus ostreatus*. Al introducir datos de una especie diferente, según el tamaño de su genoma, el tiempo de ejecución variará. Sin embargo, una de las fases, puede que varíe y mucho, en diferentes ejecuciones con genomas de la misma especie. En la fase *GenomeAnalisys*, que además resulta ser la que más aporta al tiempo de ejecución total de la herramienta, según el número de lecturas o *coverage* de que se dispongan, el tiempo será mayor o menor. El número de lecturas, depende de cómo de precisa queremos que sea la secuencia genética y de cuánto dinero se disponga para la secuenciación. Estos aspectos, dependen del interés y del estudio concretos que se estén llevando a cabo en ese momento y no del desarrollador. En la tabla, se muestra un tiempo de ejecución de 5100 segundos, 1h y 25 minutos, para analizar un total de 25'5 millones de lecturas, en una configuración de 12 cpu y 16GB de memoria, que es la configuración que suelen utilizar en el equipo en sus ejecuciones del cluster.

Capítulo 6

Conclusiones y líneas futuras

Como resumen he de decir que la mayor complejidad que he encontrado en este proyecto ha sido la distancia de conocimiento que existe entre la informática y la genética. Ha sido mayor la complejidad de incorporar a mi lenguaje los términos utilizados en el ámbito de la genética, que la complejidad del sistema y su implementación. A pesar de mis conocimientos previos en la materia, trabajar e integrarme en un grupo totalmente ajeno a mi en estudios, ha supuesto un gran esfuerzo tanto por mi parte como por la del grupo de investigación. En muchas de las reuniones que hemos tenido, a la hora de explicarme cómo es el funcionamiento de los organismos, para que posteriormente pudiera implementar en código la funcionalidad que me solicitaban, he tenido que pedir aclaraciones, buscar información adicional mientras desarrollaba el código y a pesar de ello, siempre había cuestiones que escapaban a mi control.

El cambio de tutor, supuso un antes y un después en el desarrollo de mi proyecto. Con Fran desde el principio, un desarrollo a muy bajo nivel me permitió aprender mucho sobre el funcionamiento de los genes, sus interacciones, distribución en el genoma, etc. Aunque con mucho esfuerzo y a costa de avanzar más lentamente. Ya con el nuevo tutor, pude encontrar algunas herramientas ya desarrolladas, que aceleraron enormentente mi desarrollo, aunque para ello fue necesario que Raúl me ayudara en la búsqueda de estas herramientas. Aunque estaban bien documentadas para su uso, su funcionalidad era totalmente desconocida para mi, pues sus descripciones estaban orientadas a personas con conocimientos de genética, de los que yo carecía.

Este proyecto me ha permitido experimentar cómo es el trabajo en un equipo multidisciplinar. De cómo es más necesario que en ningún otro proyecto, que todas las personas participantes compartan la visión y objetivos del proyecto, así como que queden claras las distintas etapas del proyecto y sobre todo se cree un glosario común. Especial hincapié en esto último, pues a mi parecer, al definir los conceptos, se disipan dudas y se concreta de qué se está hablando pero sobre todo de qué NO se está hablando, evitando así confusiones.

El desarrollo paulatino de una herramienta dividida en secciones, genéticamente con-

sistentes y que tuvieran sentido para el grupo investigador, me permitió ir avanzando en el proyecto con la supervisión y validación por parte del grupo de investigación, a pesar de que para mi fuera indiferente que el proyecto se dividiera en unas partes u otras. Yo no tengo por que tener profundos conocimientos de genética, ni el grupo conocimientos de informática y sin embargo entre los dos poder desarrollar una herramienta plenamente funcional.

El cambio de plataforma, tanto la migración de un ordenador de escritorio al cluster, como la actualización que sufrió el propio cluster, me permitió comprobar el buen diseño que se había realizado de la herramienta y de sus divisiones. En ambos cambios, hubo que modificar algunas partes del código, pero al haber encapsulado las funcionalidades de forma coherente, me permitió hacer las modificaciones de forma rápida y adaptarme fácilmente a los cambios.

Para dar continuidad a este proyecto sugiero las siguientes ideas:

- Optimización del código: siempre hay partes que pueden mejorar su ejecución en términos de menor uso de recursos o mayor velocidad de ejecución. También se puede mejorar la seguridad en los datos de entrada. Se ha dado por hecho que los ficheros de entrada son consistentes y cumplen con las especificaciones de cada uno de ellos, pero podría no ser siempre así.
- Paralelismo: algunas partes del código, se han implementado teniendo en cuenta que pueden ejecutarse de forma paralela, pero la ejecución en el cluster de la herramienta no. Está preparada para la ejecución de varios análisis, pero de forma secuencial.
- Ampliación de las funciones: incorporar funciones que faciliten la labor de investigación al grupo, estén relacionadas o no con este proyecto.

Capítulo 7

Anexos

Los siguientes anexos están disponibles en el CD dónde se entrega esta documentación.

1. Scripts en Python para ejecución en local
2. *Leeme* con ejemplos de ejecución y explicación de parámetros en las llamadas a los scripts
3. Datos de prueba: .fasta y .gff/.gtf de PC15 y PC9 así como .bam de N001

Bibliografía

[1] Harvey Lodish. *BIOLOGÍA CELULAR Y MOLECULAR.* ed. Médica Panamericana 2005

[2] M. Cooper, Robert E. Hausman. *LA CÉLULA* ed. Marbán 2010

[3] José Luque Cabrera, Ángel Herráez Sánchez *BIOLOGÍA MOLECULAR E INGENIERÍA GENÉTICA* ed. Elsevier España 2001

[4] Bruce Alberts. *BIOLOGÍA MOLECULAR DE LA CÉLULA* ed. Omega 2004

[5] *h*ttp://www.tecmint.com/the-truth-of-python-and-perl-features-pros-and-cons-discussed/

[6] *h*ttp://www.garshol.priv.no/download/text/perl.html

[7] *h*ttp://www.ibm.com/developerworks/ssa/library/l-perl-2-python/index.html

[8] *h*ttp://genome.sph.umich.edu/

[9] *h*ttp://samtools.github.io/hts-specs/SAMv1.html

[10] *h*ttp://www.scipy.org/

[11] *h*ttps://www.ncbi.nlm.nih.gov/pubmed/25063299

[12] *h*ttp://www.rna-seqblog.com/htseq-a-python-framework-to-work-with-high-throughput-sequencing-data/

[13] *h*ttp://www.ensembl.org/index.html

[14] *h*ttp://genome.jgi.doe.gov/pages/search-for-genes.jsf

[15] *h*ttp://www.extremeprogramming.org

[16] *h*ttp://www.bioinformaticos.com.ar/una-comparacion-de-lenguajes-de-programacion-usados-en-bioinformatica/

[17] *h*ttp://www.bioetica.org

[18] *h*ttp://biopython.org/